

BY EXPRESS MAIL NO. EL387309184US
Attorney Docket No. KOIK-P9784

MENU

SEARCH

INDEX

1/1



JAPANESE PATENT OFFICE

PATENT ABSTRACTS OF JAPAN

(11)Publication number: 05342012

(43)Date of publication of application: 24.12.1993

(51)Int.Cl.

G06F 9/45
G06F 15/347

(21)Application number: 04176273

(71)Applicant:

SONY CORP

(22)Date of filing: 10.06.1992

(72)Inventor:

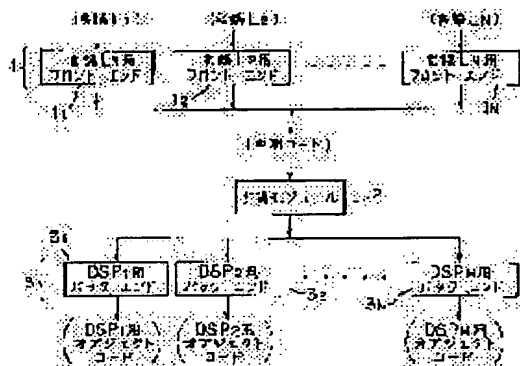
IWATA EIJI

(54) COMPILING METHOD AND COMPILER

(57)Abstract:

PURPOSE: To compile a program without depending on digital signal processors(DSP) and languages.

CONSTITUTION: At a front end part in for language Ln, corresponding to the language Ln [(n)=1, 2...N] preprocessing is performed to the source code of the program written in the language LN, and an intermediate code common for languages L1-Ln is outputted. At a common module part 2, processing is performed to the intermediate code without depending on the languages L1-LN and DSP1-DSPM. At a back end part 3M [(m)=1, 2...M] for DSPM, post-processing is performed to the output of the common module part 2, and an object code is outputted corresponding to the DSPM.



LEGAL STATUS

[Date of request for examination]
[Date of sending the examiner's decision of rejection]
[Kind of final disposal of application other than the
examiner's decision of rejection or application converted
registration]
[Date of final disposal for application]
[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of
rejection]
[Date of extinction of right]

Copyright (C); 1998 Japanese Patent Office

MENU

SEARCH

INDEX

(19)日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開平5-342012

(43)公開日 平成 5 年(1993)12月24日

(51)Int.Cl.⁵

G 0 6 F 9/45
15/347

識別記号

庁内整理番号

G 8320-5L
9292-5B

F I

G 0 6 F 9/ 44

3 2 2 A

技術表示箇所

審査請求 未請求 請求項の数 5 (全 11 頁)

(21)出願番号

特願平4-176273

(22)出願日

平成 4 年(1992) 6 月10日

(71)出願人 000002185

ソニー株式会社

東京都品川区北品川 6 丁目 7 番35号

(72)発明者 岩田 英次

東京都品川区北品川 6 丁目 7 番35号 ソニ
ー株式会社内

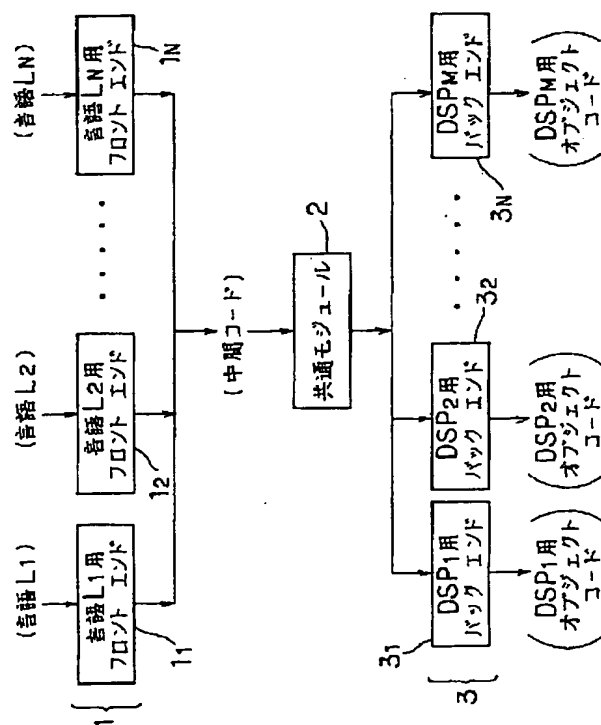
(74)代理人 弁理士 稲本 義雄 (外 1 名)

(54)【発明の名称】 コンパイル方法およびコンパイラ

(57)【要約】

【目的】 デジタルシグナルプロセッサおよび言語に依存せずに、プログラムをコンパイルする。

【構成】 言語 L_n 用フロントエンド部 1_n において、言語 L_n ($n=1, 2, \dots, N$) で記述されたプログラムのソースコードに対して、言語 L_n に対応した前処理が行われ、言語 L_1 乃至 L_N に共通の中間コードが出力される。共通モジュール部 2 において、その中間コードに対して、言語 L_1 乃至 L_N および DSP_1 乃至 DSP_M に依存しない処理が行われ、デジタルシグナルプロセッサ DSP_m 用バックエンド部 3_m ($m=1, 2, \dots, M$) において、共通モジュール部 2 の出力に対して後処理が施されて、デジタルシグナルプロセッサ DSP_m に対応したオブジェクトコードが出力される。



【特許請求の範囲】

【請求項1】 デジタルシグナルプロセッサのプログラムのソースコードを中間コードに変換し、前記中間コードを前記デジタルシグナルプロセッサに対応したオブジェクトコードに変換するコンパイル方法において、前記中間コードは、前記デジタルシグナルプロセッサのプログラムを記述するための複数の言語、または複数のデジタルシグナルプロセッサにそれぞれ対応したオブジェクトコードに共通であることを特徴とするコンパイル方法。

【請求項2】 デジタルシグナルプロセッサのプログラムのソースコードを中間コードに変換する前処理手段と、前記前処理手段より出力される中間コードを解析して最適化する共通処理手段と、前記共通処理手段からの出力を前記デジタルシグナルプロセッサに対応したオブジェクトコードに変換する後処理手段とを備え、前記前処理手段は、前記デジタルシグナルプロセッサのプログラムを記述するための複数の言語に共通の中間コードを出力し、前記後処理手段は、前記中間コードを複数のデジタルシグナルプロセッサにそれぞれ対応したオブジェクトコードに変換することを特徴とするコンパイラ。

【請求項3】 前記共通処理手段は、前記中間コードに対して、前記複数のデジタルシグナルプロセッサのどれにも依存しない最適化処理を施すことを特徴とする請求項2に記載のコンパイラ。

【請求項4】 前記後処理手段は、前記中間コードを並列に実行可能な単位に分割し、前記デジタルシグナルプロセッサに割り当てることを特徴とする請求項2または3に記載のコンパイラ。

【請求項5】 前記後処理手段は、レジスタ割付を行うことを特徴とする請求項2、3、または4に記載のコンパイラ。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、例えば音声処理や画像処理などのデジタル信号処理に用いられるDSP（デジタルシグナルプロセッサ）のプログラムをコンパイルする場合に用いて好適なコンパイル方法並びにコンパイラに関する。

【0002】

【従来の技術】近年の音声処理や画像処理などにおいては、多くの場合、信号（音声信号や画像信号）をサンプリングし、即ち信号をデジタル化し、例えばDFT（離散フーリエ変換）、FFT（高速フーリエ変換）、またはDCT（離散コサイン変換）などにより、時間軸上の信号を周波数軸上の信号（スペクトル）に変換し

て、パラメータの抽出や圧縮符号化処理が行われる。

【0003】このような、デジタル信号の時間軸と周波数軸との相互変換（DFT（逆DFT）、FFT（逆FFT）、およびDCT（逆DCT）など）処理に代表されるデジタル信号処理においては、積和演算の回数が非常に多く、この処理を、例えば汎用的なCPUなどで実現した場合には、実時間処理を行うことが困難であった。

【0004】そこで、いわゆるパイプライン処理や並列処理に適したアーキテクチャを有し、積和演算を、例えば数10乃至数100n秒で行うことのできる、いわば積和演算処理を得意とするDSP（デジタルシグナルプロセッサ）が開発され、近年、このDSPを用いた多くのデジタル信号処理装置（例えば画像処理装置や音声処理装置など）が実現されている。

【0005】

【発明が解決しようとする課題】ところで、このDSPに、デジタル信号処理を実行させるには、まず、そのDSP用の言語でデジタル信号処理プログラムを記述し、そのソースコードをコンパイラでコンパイルしてオブジェクトコードに変換する必要がある。

【0006】しかしながら、デジタル信号処理プログラムを記述するための言語は、DSPにより異なる場合が多く、また同じDSPであってもプログラムを記述するための言語が複数種類提供されている場合もあり、従って、例えばM個のDSPのプログラムが、N種類の言語で記述することができるときには、各DSPおよび各言語に対応したM×N個のコンパイラを用意しなければならず、不便であった。

【0007】なお、言語がDSPにより異なる場合とは、言語の種類がDSPにより異なる場合の他、同じ言語であってもその系統がDSPにより異なる場合（例えば、コンピュータ言語でいえば、同じC言語でも、コンパイラの提供メカにより異なる場合）を含む。

【0008】さらに、コンパイラは、プログラムをコンパイルする過程を分割した、複数の論理的な操作単位（フェーズ）よりなっているが、上述した各DSPおよび各言語に対応した各コンパイラには、DSPおよび言語に依存しない、互いに共用することができるフェーズが、少なからず含まれており、各コンパイラにこのフェーズを設けることは無駄であった。

【0009】本発明は、このような状況に鑑みてなされたものであり、DSPおよび言語に依存せずに、プログラムをコンパイルすることができるようにするものである。

【0010】

【課題を解決するための手段】請求項1に記載のコンパイル方法は、デジタルシグナルプロセッサのプログラムのソースコードを中間コードに変換し、中間コードをデジタルシグナルプロセッサに対応したオブジェクト

コードに変換するコンパイル方法において、中間コードは、デジタルシグナルプロセッサのプログラムを記述するための複数の言語、または複数のデジタルシグナルプロセッサにそれぞれ対応したオブジェクトコードに共通であることを特徴とする。

【0011】請求項2に記載のコンパイラは、デジタルシグナルプロセッサDSP₁乃至DSP_Mのプログラムのソースコードを中間コードに変換する前処理手段としてのフロントエンド部1（言語L₁用フロントエンド部1₁乃至言語L_N用フロントエンド部1_N）と、フロントエンド部1より出力される中間コードを解析して最適化する共通処理手段としての共通モジュール部2と、共通モジュール部2からの出力をデジタルシグナルプロセッサDSP₁乃至DSP_Mに対応したオブジェクトコードに変換する後処理手段としてのバックエンド部3（DSP₁用バックエンド部3₁乃至DSP_M用バックエンド部3_M）とを備え、フロントエンド部1（言語L₁用フロントエンド部1₁乃至言語L_N用フロントエンド部1_N）は、デジタルシグナルプロセッサDSP₁乃至DSP_Mのプログラムを記述するための複数の言語L₁乃至L_Nに共通の中間コードを出力し、バックエンド部3（DSP₁用バックエンド部3₁乃至DSP_M用バックエンド部3_M）は、中間コードを複数のデジタルシグナルプロセッサDSP₁乃至DSP_Mにそれぞれ対応したオブジェクトコードに変換することを特徴とする。

【0012】請求項3に記載のコンパイラは、共通モジュール部2に、中間コードに対して、DSP₁乃至DSP_Mのどれにも依存しない最適化処理を施させることを特徴とする。

【0013】請求項4に記載のコンパイラは、バックエンド部3（DSP₁用バックエンド部3₁乃至DSP_M用バックエンド部3_M）に、中間コードを並列に実行可能な単位に分割させ、DSP₁乃至DSP_Mに割り当てさせることを特徴とする。

【0014】請求項5に記載のコンパイラは、バックエンド部3（DSP₁用バックエンド部3₁乃至DSP_M用バックエンド部3_M）に、レジスタ割付を行わせることを特徴とする。

【0015】

【作用】請求項1に記載のコンパイル方法においては、デジタルシグナルプロセッサのプログラムのソースコードを、デジタルシグナルプロセッサのプログラムを記述するための複数の言語、または複数のデジタルシグナルプロセッサにそれぞれ対応したオブジェクトコードに共通の中間コードに変換し、その中間コードをデジタルシグナルプロセッサに対応したオブジェクトコードに変換する。従って、DSP（デジタルシグナルプロセッサ）および言語に依存せずに、プログラムをコンパイルすることができる。

【0016】請求項2に記載のコンパイラにおいては、

DSP₁乃至DSP_Mのプログラムのソースコードを言語L₁乃至L_Nに共通の中間コードに変換し、その中間コードを解析して最適化する。そして、最適化した中間コードをDSP₁乃至DSP_Mにそれぞれ対応したオブジェクトコードに変換する。従って、DSP（デジタルシグナルプロセッサ）および言語に依存せずに、プログラムをコンパイルすることができる。

【0017】請求項3に記載のコンパイラにおいては、共通モジュール部2に、中間コードに対して、DSP₁乃至DSP_Mのどれにも依存しない最適化処理を施させるので、重複したフェーズによる無駄な処理が防止される。

【0018】請求項4に記載のコンパイラは、バックエンド部3（DSP₁用バックエンド部3₁乃至DSP_M用バックエンド部3_M）に、中間コードを並列に実行可能な単位に分割させ、DSP₁乃至DSP_Mに割り当てさせるので、並列処理をすることができるDSP（デジタルシグナルプロセッサ）の機能を有効に利用することができる。

【0019】請求項5に記載のコンパイラは、バックエンド部3（DSP₁用バックエンド部3₁乃至DSP_M用バックエンド部3_M）に、レジスタ割付を行わせるので、DSP（デジタルシグナルプロセッサ）のレジスタを効率的に利用することができる。

【0020】

【実施例】図1は、本発明のコンパイラの一実施例の構成を示すブロック図である。フロントエンド部1は、M個のデジタルシグナルプロセッサDSP₁乃至DSP_M（以下、DSP₁乃至DSP_M）のいずれかのプログラムの記述用言語L₁乃至L_Nで記述されたプログラムのソースコードを、言語L₁乃至L_Nに共通の中間言語で記述される中間コードにそれぞれ変換する言語L₁用フロントエンド部1₁乃至言語L_N用フロントエンド部1_Nから構成される。各言語L_n用フロントエンド部1_n（n=1, 2, ..., N）は、図2に示すフェーズ（コンパイルの処理単位）としての、字句解析部11_n、構文解析部12_n、および意味解析部13_nから構成され、言語L_nに依存した処理を行う。

【0021】即ち、言語L_n用フロントエンド部1_nの字句解析部11_nは、言語L_nで記述されたプログラムのソースコードを、論理的に扱うことのできる最小単位の文字列（字句）に分離する。構文解析部12_nは、字句解析部11_nより出力される字句からなる構文を構造解析し、言語L_nの構文構造として許されているか否かを検査（構文チェック）する。意味解析部13_nは、字句からなる構文を意味解析し、構文の意味上の間違いをチェックし、間違いがなければソースコードを、言語L₁乃至L_Nのすべてに共通の中間コードに変換する。

【0022】共通モジュール部2は、図3に示すフェーズとしての依存解析部21および最適化部22より構成

され、言語 L_1 用フロントエンド部 1_1 乃至言語 L_N 用フロントエンド部 1_N から出力される中間コードに対して、言語 L_1 乃至 L_N および DSP_1 乃至 DSP_M に依存しない処理を行う。

【0023】即ち、共通モジュール部2の依存解析部21は、フロントエンド部1より出力される中間コードにおける、データの参照または定義の依存関係（データ依存）を解析し、この依存関係（データ依存）を、いわゆるデータフローグラフと呼ばれるデータ構造に変換する。最適化部22は、必要に応じて依存解析部21で作成されたデータフローグラフを参照し、フロントエンド部1より出力される中間コードに対して、例えば定数の畳み込みや共通部分式の識別など、言語 L_1 乃至 L_N および DSP_1 乃至 DSP_M に依存しない最適化処理を施す。

【0024】バックエンド部3は、共通モジュール部2で最適化された中間コードから、 DSP_1 乃至 DSP_M に対応したオブジェクトコードをそれぞれ生成する DSP_1 用バックエンド部 3_1 乃至 DSP_M 用バックエンド部 3_M から構成される。各 DSP_m 用バックエンド部 3_m ($m=1, 2, \dots, M$) は、図4に示すフェーズとしての、スケジューリング部 3_{1m} 、コード生成/レジスタ割付部 3_{2m} 、および最適化部 3_{3m} から構成され、 DSP_m に依存した処理を行う。

【0025】即ち、スケジューリング部 3_{1m} は、例えば DSP_m のアーキテクチャが並列処理を行うことができるものである場合、共通モジュール部2で最適化された中間コードを、並列に実行可能な単位（タスク）に分割し、 DSP_m を構成する、例えばALU、乗算器、加算器、またはシフトなどに割り当てる。コード生成/レジスタ割付部 3_{2m} は、共通モジュール部2（最適化部22）より出力される中間コードから、 DSP_m に対応したオブジェクトコードを生成するとともに、 DSP_m の有するレジスタに、効率的に変数を割り付けるためのレジスタ割付を行う。最適化部 3_{3m} は、必要に応じて共通モジュール部2の依存解析部21で作成されたデータフローグラフを参照し、コード生成/レジスタ割付部 3_{2m} で生成された DSP_m に対応したオブジェクトコードに対して、 DSP_m に依存した最適化処理を施す。

【0026】このように構成されるコンパイラにおいて、言語 L_n で記述された DSP_m 用のプログラムがコンパイルされる場合、まずフロントエンド部1の言語 L_n 用フロントエンド部 1_n にそのプログラムが読み込まれる。そして、言語 L_n 用フロントエンド部 1_n の字句解析部 1_{1n} （図2）において、言語 L_n で記述されたプログラムのソースコードが、例えば手掛かり語（例えば、C

$$A = B + C$$

$$B = A + E$$

$$B = F + G$$

という3つの式が、式（1）、（2）、（3）の順番で実行されるように、プログラムが記述されているとする

言語やFORTRANでいうところのdo, while, if、およびforなど）、識別子（例えば、プログラムにおける変数など）、定数、並びに演算子（例えば、+、-、*、および/など）など、論理的に扱うことのできる最小単位の文字列（字句）に分離される。

【0027】即ち、字句解析部 1_{1n} において、例えば
IF (5. EQ. MAX) GOTO 100
というFORTRAN文は、

10 IF, (, 5, . EQ. , MAX,), GOTO, および100
という8つの字句に分離される。

【0028】字句解析部 1_{1n} で分離された字句は構文解析部 1_{2n} に入力され、構文解析部 1_{2n} において、この字句からなる構文が構造解析され、言語 L_n の構文構造として許されているか否かが検査（構文チェック）される。即ち、例えば字句A、+、およびBが入力された場合、構文解析部 1_{2n} において、字句A、+、およびBからなる構文 $A+B$ が、式と名付ける構文構造を有すると構造解析され、構文 $A+B$ が言語 L_n で記述された式として許されているか否かが検査（構文チェック）される。構文解析部 1_{2n} で、任意の構文が、構文チェックで使用不可と判定された場合には、コンパイル（以後の処理）が中止される。

【0029】構文解析部 1_{2n} での構文チェックをパスした構文は、意味解析部 1_{3n} で、意味解析され、構文の意味上の間違いがチェックされる。即ち、意味解析部 1_{3n} において、例えば整数型の変数A、演算子+、および実数型の変数Bの3つの字句からなる構文 $A+B$ が、整数型と実数型の加算式であると解析され、整数と実数との加算が間違いか否かがチェックされる。言語 L_n の仕様で、整数と実数との加算が許されていなければ、意味解析部 1_{3n} で、構文 $A+B$ は使用不可とされ、コンパイル（以後の処理）が中止される。

【0030】意味解析部 1_{3n} で、構文の意味上の間違いがチェックされた構文に間違いがなければ、ソースコードが、言語 L_1 乃至 L_N のすべてに共通の中間コードに変換され、共通モジュール部2の依存解析部21（図3）に出力される。

【0031】依存解析部21において、意味解析部 1_{3n} より出力された中間コードにおける、データの参照または定義の依存関係（データ依存）が解析され、この依存関係（データ依存）が、いわゆるデータフローグラフと呼ばれるデータ構造に変換される。

【0032】即ち、例えば

$$(1)$$

$$(2)$$

$$(3)$$

と、依存解析部21では、

50 ・式（1）で定義されたAが式（2）で参照されている

(フロー依存)。

・式(1)で参照されたBが式(2)で定義されている(逆依存)。

・式(1)で参照されたBが式(3)で定義されている(逆依存)。

・式(2)で定義されたBが式(3)で再定義されている(出力依存)。

のように依存解析がなされ、このデータの参照または定義の依存関係(データ依存)がいわゆるデータフローグラフと呼ばれるデータ構造に変換される。

【0033】依存解析部21でデータフローグラフが作成された後、最適化部22において、必要に応じてこのデータフローグラフが参照され、意味解析部13_nより出力された中間コードに対して、例えば定数の畳み込みや共通部分式の識別など、DSP_m乃至DSP_mに依存しない最適化処理が施される。

【0034】即ち、例えば

$I = 4$

$A = I * B$

という構文がプログラムの中に言語L_nで記述されると、最適化部22において、上記の構文は、

$A = 4 * B$

と最適化され、これにより定義($I = 4$)の回数を減らすことができる。

【0035】また、例えば

$A[I+1] = B[I+1] * C[I+1]$

いう構文がプログラムの中に言語L_nで記述されていると、最適化部22において、上記の構文は、

$J = I + 1$

$y(0) = h(0) * x(0) + h(1) * x(1) + h(2) * x(2)$

$y(1) = h(0) * x(1) + h(1) * x(2) + h(2) * x(3)$

$y(2) = h(0) * x(2) + h(1) * x(3) + h(2) * x(4)$

を例にして、DSP_m用バックエンド部3_mのスケジューリング部31_m、コード生成/レジスタ割付部32_m、および最適化部33_mの動作を、さらに説明する。なお、このプログラムにおいて、 $h(0)$ 、 $h(1)$ 、および $h(2)$ はフィルタの係数、 $x(0)$ 、 $x(1)$ 、...

$temp1 = h(0) * x(0)$ (a1)

$temp2 = h(1) * x(1)$ (a2)

$temp3 = h(2) * x(2)$ (a3)

$temp4 = temp1 + temp2$ (a4)

$y(0) = temp3 + temp4$ (a5)

$temp5 = h(0) * x(1)$ (b1)

$temp6 = h(1) * x(2)$ (b2)

$temp7 = h(2) * x(3)$ (b3)

$temp8 = temp5 + temp6$ (b4)

$y(1) = temp7 + temp8$ (b5)

$temp9 = h(0) * x(2)$ (c1)

$temp10 = h(1) * x(3)$ (c2)

$temp11 = h(2) * x(4)$ (c3)

$A[J] = B[J] * C[J]$

と最適化され、これにより加算($I+1$)の回数を3回から1回に減らすことができる。

【0036】共通モジュール部2の最適化部22で最適化された中間コードは、DSP_m用バックエンド部3_mのスケジューリング部31_m(図4)に出力され、スケジューリング部31_mにおいて、DSP_mのアーキテクチャが並列処理を行うことができるものである場合には、共通モジュール部2で最適化された中間コードが、並列に実行可能な単位(タスク)に分割され、分割された各タスクが、DSP_mを構成する、例えばALU、乗算器、加算器、またはシフトなどに割り当てられる(スケジューリングされる)。

【0037】そして、コード生成/レジスタ割付部32_mにおいて、共通モジュール部2の最適化部22より出力された中間コードから、DSP_mに対応したオブジェクトコードが生成されるとともに、DSP_mの有するレジスタに、効率的に変数を割り付けるためのレジスタ割付が行われ、DSP_mに対応したオブジェクトコードが最適化部33_mに出力される。最適化部33_mにおいて、必要に応じて共通モジュール部2の依存解析部21で作成されたデータフローグラフが参照され、コード生成/レジスタ割付部32_mより出力されたオブジェクトコードに対して、DSP_mに依存した最適化処理が施され、言語L_nで記述されたDSP_m用のプログラムのコンパイルが終了する。

【0038】以下、DSP_mで3タップのFIR型ディジタルフィルタを実現するプログラム

・はフィルタへの入力信号、 $y(0)$ 、 $y(1)$ 、...

・はフィルタ出力を示す。

【0039】上記のソースコードを中間コードで表現すると、

temp12=temp9+temp10
y(2)=temp11+temp12

(c 4)

(c 5)

となる。

【0040】ここで、DSP_mが、充分な数のレジスタを有するとすると、式(a 1)乃至(a 5)、式(b 1)乃至(b 5)、および式(c 1)乃至(c 5)（以下、式(a 1)乃至(c 5)と記載する）のデータ依存は、前述したフロー依存のみとなり、式(a 1)および式(a 2)での定義が、式(a 4)で参照されているというフロー依存を

(a 1), (a 2) → (a 4)

と表すと、式(a 1)乃至(a 5)、式(b 1)乃至(b 5)、および式(c 1)乃至(c 5)のデータ依存は、

(a 1), (a 2) → (a 4)

(a 3), (a 4) → (a 5)

(b 1), (b 2) → (b 4)

(b 3), (b 4) → (b 5)

(c 1), (c 2) → (c 4)

(c 3), (c 4) → (c 5)

となる。

【0041】DSP_mが並列に実行可能なアーキテクチャを有さない場合、即ち例えば演算器として2入力1出力の演算器を1つだけDSP_mが有する場合、スケジューリング部31_mにおいて、式(a 1)乃至(c 5)で示される演算が、DSP_mが内蔵する1つだけの演算器に、図5に示すように割り当てられる（スケジューリングされる）。

【0042】即ち、DSP_mが内蔵する演算器が1クロックで動作するとすると、スケジューリング部31_mでは、式(a 1)乃至(c 5)で示される演算が、それぞれ1乃至15クロック目に、演算器で行われるように割り当てられる。

【0043】また、DSP_mが並列に実行可能なアーキテクチャを有する場合、即ち例えば演算器として、9個の、2入力1出力の乗算器X₁乃至X₉、および6個の、2入力1出力の加算器Y₁乃至Y₆をDSP_mが有する場合、スケジューリング部31_mにおいて、式(a 1)乃至(c 5)で示される演算が、DSP_mが内蔵する乗算器X₁乃至X₉および加算器Y₁乃至Y₆に、図6に示すように割り当てられる（スケジューリングされる）。

【0044】即ち、乗算器X₁乃至X₉および加算器Y₁乃至Y₆が1クロックで動作するとすると、スケジューリング部31_mにおいて、1クロック目に、式(a 1)乃至(a 3)、式(b 1)乃至(b 3)、または式(c 1)乃至(c 3)で示される演算（乗算）が、乗算器X₁乃至X₉でそれぞれ行われるように割り当てられ、2クロック目に、式(a 4)、(b 4)、または(c 4)で示される演算（加算）が、加算器Y₁、Y₃、またはY₅でそれぞれ行われるように割り当てられるとともに、3

クロック目に、式(a 5)、(b 5)、または(c 5)で示される演算（加算）が、加算器Y₂、Y₄、またはY₆でそれぞれ行われるように割り当てられる。

【0045】さらに、DSP_mが、例えば演算器として、3個の、2入力1出力の乗算器X₁乃至X₃、並びに2個の、2入力1出力の加算器Y₁およびY₂を有する場合、スケジューリング部31_mにおいて、式(a 1)乃至(c 5)で示される演算が、DSP_mが内蔵する乗算器X₁乃至X₃並びに加算器Y₁およびY₂に、図7に示すように割り当てられる（スケジューリングされる）。

【0046】即ち、乗算器X₁乃至X₃並びに加算器Y₁およびY₂が1クロックで動作するとすると、スケジューリング部31_mにおいて、1クロック目に、式(a 1)乃至(a 3)で示される演算が、乗算器X₁乃至X₃でそれぞれ行われるように割り当てられ、2クロック目に、式(b 1)乃至(b 3)、または式(a 4)で示される演算が、乗算器X₁乃至X₃、または加算器Y₁でそれぞれ行われるように割り当てられるとともに、3クロック目に、式(c 1)乃至(c 3)、式(b 4)、または式(a 5)で示される演算が、乗算器X₁乃至X₃、加算器Y₁またはY₂でそれぞれ行われるように割り当てられる。さらに、4クロック目には、式(c 4)または式(b 5)で示される演算が、加算器Y₁またはY₂でそれぞれ行われるように割り当てられ、5クロック目に、式(c 5)で示される演算が、加算器Y₂で行われるように割り当てられる。

【0047】また、DSP_mが、例えば演算器として、2個の、2入力1出力の乗算器X₁およびX₂、並びに1個の、2入力1出力の加算器Y₁を有する場合、スケジューリング部31_mにおいて、式(a 1)乃至(c 5)で示される演算が、DSP_mが内蔵する乗算器X₁およびX₂並びに加算器Y₁に、図8に示すように割り当てられる（スケジューリングされる）。

【0048】即ち、乗算器X₁およびX₂並びに加算器Y₁が1クロックで動作するとすると、スケジューリング部31_mにおいて、乗算器X₁で、式(a 1)、(a 3)、(b 2)、(c 1)、または(c 3)で示される演算が、1乃至5クロック目にそれぞれ行われるように割り当てられ、乗算器X₂で、式(a 2)、(b 1)、(b 3)、または(c 2)で示される演算が、1乃至4クロック目にそれぞれ行われるように割り当てられるとともに、加算器Y₁で、式(a 4)、(a 5)、(b 4)、(b 5)、(c 4)、または(c 5)で示される演算が、2乃至7クロック目にそれぞれ行われるように割り当てられる。

【0049】スケジューリング部31_mでのスケジューリングが終了すると、コード生成/レジスタ割付部32_mにおいて、DSP_mの有するレジスタに、効率的に変数

を割り付けるためのレジスタ割付が行われながら、式 (a 1) 乃至式 (c 5) に示す中間コードから、DSP_mに対応したオブジェクトコードが生成される。

【0050】DSP_mが3つのレジスタ reg 1 乃至 reg 3 を有する場合、コード生成／レジスタ割付部 32

MUL h (0), x (0), reg 1	(A 1)
MUL h (1), x (1), reg 2	(A 2)
MUL h (2), x (2), reg 3	(A 3)
ST reg 3, mem 1	(A 4)
ADD reg 1, reg 2, reg 3	(A 5)
LD mem 1, reg 1	(A 6)
ADD reg 1, reg 3, y (0)	(A 7)
MUL h (0), x (1), reg 1	(B 1)
MUL h (1), x (2), reg 2	(B 2)
MUL h (2), x (3), reg 3	(B 3)
ST reg 3, mem 2	(B 4)
ADD reg 1, reg 2, reg 3	(B 5)
LD mem 2, reg 1	(B 6)
ADD reg 1, reg 3, y (1)	(B 7)
MUL h (0), x (2), reg 1	(C 1)
MUL h (1), x (3), reg 2	(C 2)
MUL h (2), x (4), reg 3	(C 3)
ST reg 3, mem 3	(C 4)
ADD reg 1, reg 2, reg 3	(C 5)
LD mem 3, reg 1	(C 6)
ADD reg 1, reg 3, y (2)	(C 7)

のように生成される。

【0051】オブジェクトコード (A 1) 乃至 (A 3) は、

reg 1 = h (0) * x (0),
reg 2 = h (1) * x (1)、または
reg 3 = h (2) * x (2)

をそれぞれ示し、式 (a 1) 乃至 (a 3) に対応する。DSP_mには、3つのレジスタ reg 1 乃至 reg 3 しかないので、式 (a 4) を計算するために、オブジェクトコード (A 4) で、レジスタ reg 3 の内容 (式 (a 3) の変数 temp 3 の内容) がスタック mem 1 に退避され (ストアされ)、オブジェクトコード (A 5) で、式 (a 4) に対応する
reg 3 = reg 1 + reg 2
が計算される。

【0052】そして、オブジェクトコード (A 6) で、スタック mem 1 にストアされた式 (a 3) の変数 temp 3 の内容 (h (2) * x (2)) が、レジスタ reg 1 にロードされ、オブジェクトコード (A 7) で、式 (a 5) に対応する
y (0) = reg 1 + reg 3
が計算される。

【0053】オブジェクトコード (B 1) 乃至 (B 7)、または (C 1) 乃至 (C 7) でも、上記したオブ

ジェクトコード (A 1) 乃至式 (c 5) における変数 temp 1 乃至 temp 12 が、レジスタ reg 1 乃至 reg 3 に割り付けられながら、式 (a 1) 乃至式 (c 5) で示された中間コードから、DSP_mに対応したオブジェクトコードが、例えば

(A 1)
(A 2)
(A 3)
(A 4)
(A 5)
(A 6)
(A 7)
(B 1)
(B 2)
(B 3)
(B 4)
(B 5)
(B 6)
(B 7)
(C 1)
(C 2)
(C 3)
(C 4)
(C 5)
(C 6)
(C 7)

ジェクトコード (A 1) 乃至 (A 7) における場合と同様に、式 (b 1) 乃至 (b 5)、または (c 1) 乃至 (c 5) にそれぞれ対応する処理がなされる。

30 【0054】なお、DSP_mが、例えば1クロックで動作する演算器として、3個の、2入力1出力の乗算器 X₁ 乃至 X₃、並びに2個の、2入力1出力の加算器 Y₁ および Y₂ を有し、データのロードおよびストアを1クロックで行うとすると、上記のオブジェクトコード (A 1) 乃至 (A 7)、(B 1) 乃至 (B 7)、および (C 1) 乃至 (C 7) は、図9に示すように実行される。

【0055】即ち、1クロック目に、オブジェクトコード (A 1) 乃至 (A 3) に対応する式 (a 1) 乃至 (a 3) で示される演算が、乗算器 X₁ 乃至 X₃ でそれぞれ行われ、2クロック目に、オブジェクトコード (B 1) 乃至 (B 3) に対応する式 (b 1) 乃至 (b 3) で示される演算が、乗算器 X₁ 乃至 X₃ でそれぞれ行われるとともに、オブジェクトコード (A 4) に対応するレジスタ reg 3 のスタック mem 1 へのストアが行われる。

40 【0056】3クロック目に、オブジェクトコード (C 1) 乃至 (C 3) または (A 5) に対応する式 (c 1) 乃至 (c 3)、または式 (a 4) で示される演算が、乗算器 X₁ 乃至 X₃、加算器 Y₁ でそれぞれ行われるとともに、オブジェクトコード (B 4) に対応するレジスタ reg 3 のスタック mem 2 へのストアが行われ、4クロ

ーリングを説明するための図である。

【図7】図4のスケジューリング部で行われるスケジューリングを説明するための図である。

【図8】図4のスケジューリング部で行われるスケジューリングを説明するための図である。

【図9】図4のコード生成／レジスタ割付部32_mで生成されたDSP_mに対応したオブジェクトコードが実行される様子を説明するための図である。

【符号の説明】

1 フロントエンド部

1₁乃至1_N 言語L₁用フロントエンド部乃至言語L_N用フロントエンド部

2 共通モジュール部

3 バックエンド部

3₁乃至3_M DSP₁用バックエンド部乃至DSP_M用バックエンド部

11_n 字句解析部

12_n 構文解析部

13_n 意味解析部

21 依存解析部

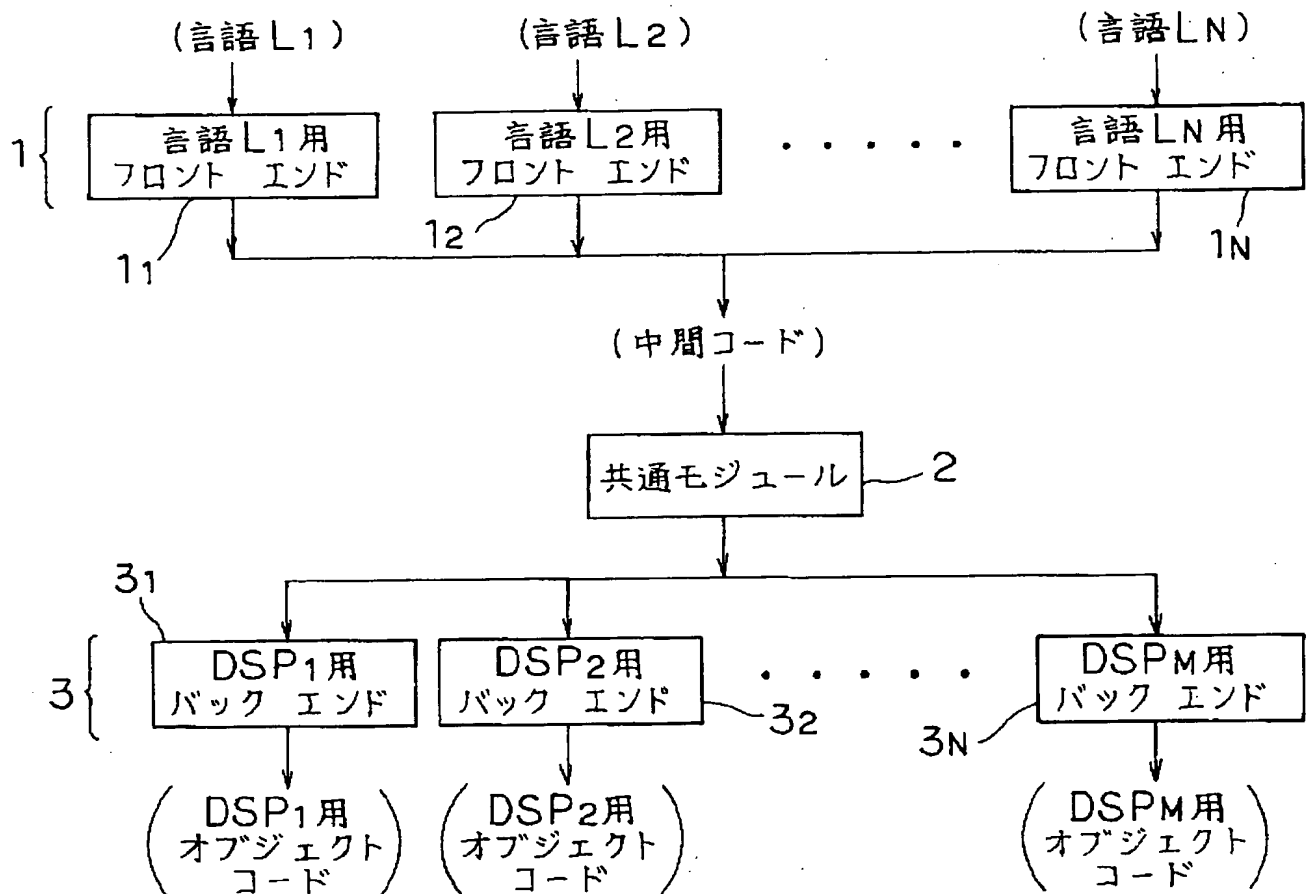
22 最適化部

10 31_m スケジューリング部

32_m コード生成／レジスタ割付部

33_m 最適化部

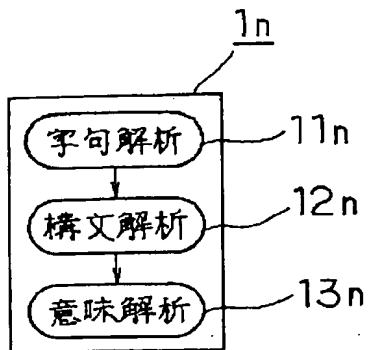
【図1】



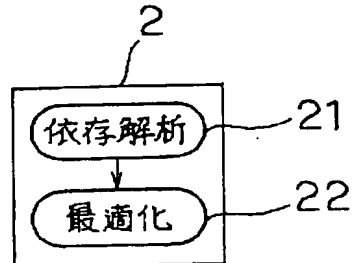
【図5】

クロック	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
演算器	(a1)	(a2)	(a3)	(a4)	(a5)	(b1)	(b2)	(b3)	(b4)	(b5)	(c1)	(c2)	(c3)	(c4)	(c5)

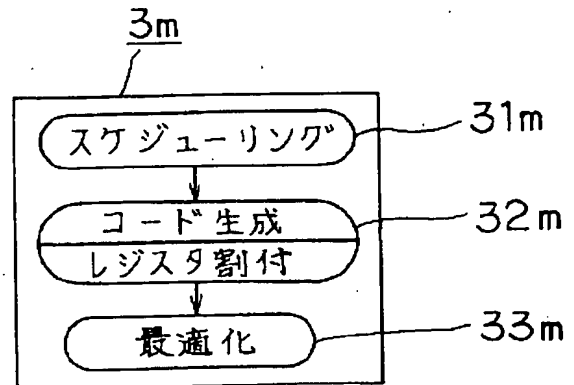
【図2】



【図3】



【図4】



【図6】

クロック	1	2	3
乗算器 X ₁	(a1)		
乗算器 X ₂	(a2)		
乗算器 X ₃	(a3)		
加算器 Y ₁		(a4)	
加算器 Y ₂			(a5)
乗算器 X ₄	(b1)		
乗算器 X ₅	(b2)		
乗算器 X ₆	(b3)		
加算器 Y ₃		(b4)	
加算器 Y ₄			(b5)
乗算器 X ₇	(c1)		
乗算器 X ₈	(c2)		
乗算器 X ₉	(c3)		
加算器 Y ₅		(c4)	
加算器 Y ₆			(c5)

【図7】

クロック	1	2	3	4	5
乗算器 X ₁	(a1)	(b1)	(c1)		
乗算器 X ₂	(a2)	(b2)	(c2)		
乗算器 X ₃	(a3)	(b3)	(c3)		
加算器 Y ₁		(a4)	(b4)	(c4)	
加算器 Y ₂			(a5)	(b5)	(c5)

【図8】

クロック	1	2	3	4	5	6	7
乗算器 X ₁	(a1)	(a3)	(b2)	(c1)	(c3)		
乗算器 X ₂	(a2)	(b1)	(b3)	(c2)			
加算器 Y ₁		(a4)	(a5)	(b4)	(b5)	(c4)	(c5)

【図9】

クロック	1	2	3	4	5	6	7
乗算器 X ₁	(A1)	(B1)	(C1)				
乗算器 X ₂	(A2)	(B2)	(C2)				
乗算器 X ₃	(A3)	(B3)	(C3)				
加算器 Y ₁			(A5)	(B5)	(C5)		
加算器 Y ₂					(A7)	(B7)	(C7)
ストア		(A4)	(B4)	(C4)			
ロード				(A6)	(B6)	(C6)	